Computação Paralela em JavaScript

Clauber P. Stipkovic H.

Professor Doutor Calebe de Paula Biachini

Apoio PIVIC Mackenzie

ABSTRACT

The objective of this scientific research have its starting point to present how a JavaScript engine works and more specifically the Mozilla's JavaScript engine called SpiderMonkey, investigate and if possible apply parallel computing concepts and techniques this JavaScript engine.

INTRODUÇAO

Nos últimos anos, a web tem se tornado incrivelmente importante tanto para a computação em geral quanto para os usuários, que interagem com vários serviços e aplicações, onde essas, acontecem consequentemente através de um web browser.

Desde a “guerra dos navegadores” em meados de 1995, entre os web browsers Microsoft Internet Explorer e Netscape Navigator, até hoje, muitas aplicações que eram conhecidas por serem tipicamente desktop, foram disponibilizadas para que pudessem ser acessadas através dos web browsers, sendo hoje em dia, muitas novas aplicações, criadas apenas para funcionar na web e em vários dispositivos como mobile, tablets e Desktops.

Para que muitas dessas aplicações se tornem possíveis nos web browsers, uma parte importante se faz necessária, o JavaScript engine.

Por verificarmos a importância de termos uma JavaScript engine que seja rápida e eficiente para gerenciar o uso da CPU dos devices onde e utilizada, esta pesquisa utilizou como base a JavaScript engine da Fundação Mozilla, chamada SpiderMonkey, onde foram realizados os estudos para entender como se da o funcionamento de uma JavaScript engine e com isso, identificar possíveis pontos onde fosse possível aplicar conceitos de computação paralela.

Tendo em vista que o SpiderMonkey foi o primeiro JavaScript engine da historia, criado para um web browser, sua importância e relevância para a web, justifica sua utilização como base para essa pesquisa.

Ressaltamos que o projeto SpiderMonkey é projeto de código aberto, o que quer dizer que o produto esta disponível para estudo, pesquisa e alteração, e que 'e possível através da Mozilla Public License version 2.0 (MPL2), com isso torna sua utilização nessa pesquisa viável.

A LINGUAGEM JAVASCRIPT

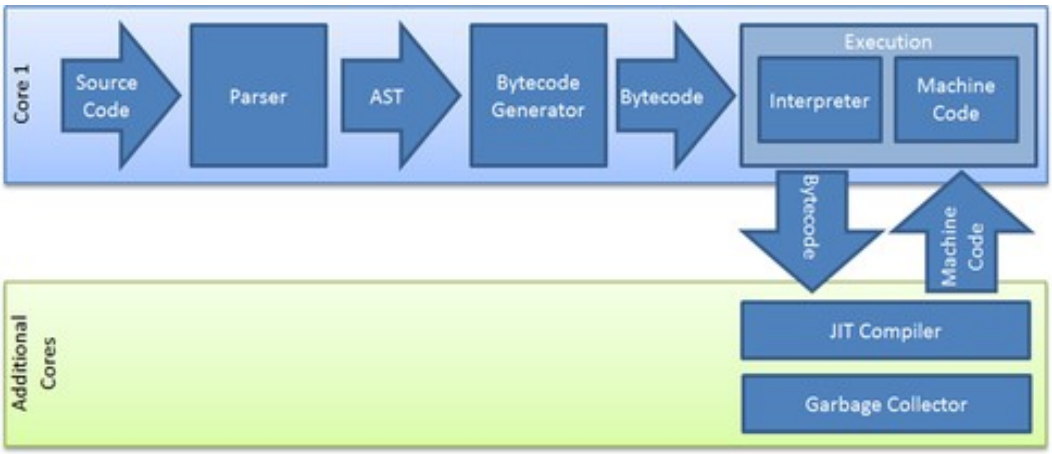
A linguagem JavaScript - também conhecida como ECMAScript - é um linguagem de programação dinâmica e multiparadigma (script, orientada a objetos, imperativa e funcional), conhecida por ser utilizada para interagir com o lado cliente dos web browser (através da plataforma Document Object Model), comunicação assíncrona e mais atualmente por também haver a possibilidade de ser executada do lado servidor, através do projeto NodeJS. Foi criada em 1995 por Brendan Eich na empresa Netscape Communications Corporation (atualmente a Fundação Mozilla) para a adoção de tecnologias web na plataforma da empresa.

DEFINIÇÃO DE UMA JAVASCRIPT ENGINE

Uma JavaScript engine – também conhecida como JavaScript interpreter ou JavaScript implementation - é uma virtual machine que interpreta e executa códigos JavaScript. Seu uso mais comum é em um web browser, mas existem outros como por exemplo, do lado do servidor ou em dispositivos moveis.

FLUXO DE FUNCIONAMENTO DA JAVASCRIPT ENGINE

O esquema teórico abaixo, mostra o fluxo que um código JavaScript percorre na JavaScript engine, desde recebimento ate a sua execução.



Depois de obter o esquema, utilizamos como exemplo a função mostrada abaixo, que faz uma soma entre dois números e retorna o resultado dessa soma.

Esse função foi utilizada como prova de conceito para mostrar passo a passo os resultados de cada etapa to processo na JavaScript engine.

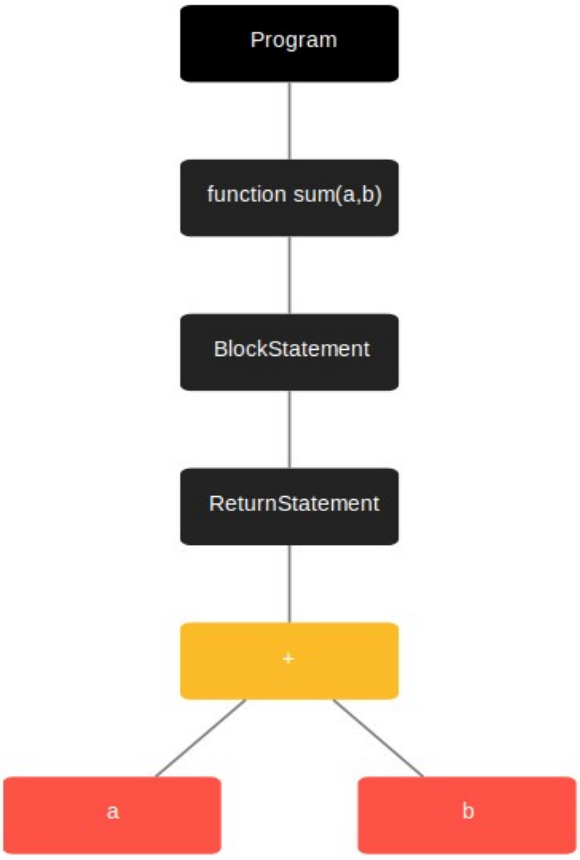
```
1 function sum(a, b) {
2     return (a + b);
3 }
4
```

Ao receber o trecho de código, a JavaScript engine inicia o parsing (também chamada de analise sintática), processo que consiste em analisar uma sequencia de símbolos em linguagem natural ou linguagem de computador, construindo uma estrutura hierárquica correspondente ao que foi informado antes do método de parsing ser aplicado.

Utilizado a função de soma como exemplo, o resultado do parsing para o trecho de código que será geral é:

```
# of nodes: 7
# of tokens: 16
Tokens:
        Keyword(function)  Identifier(sum)
        Punctuator(()  Identifier(a)  Punctuator(,)
        Identifier(b)  Punctuator())  Punctuator({)
        Keyword(return)  Punctuator(()
        Identifier(a)  Punctuator(+)  Identifier(b)
        Punctuator())  Punctuator(;)
        Punctuator(})
```

Com isso, uma abtract syntax tree (AST) é formada, como no exemplo para a função de soma:

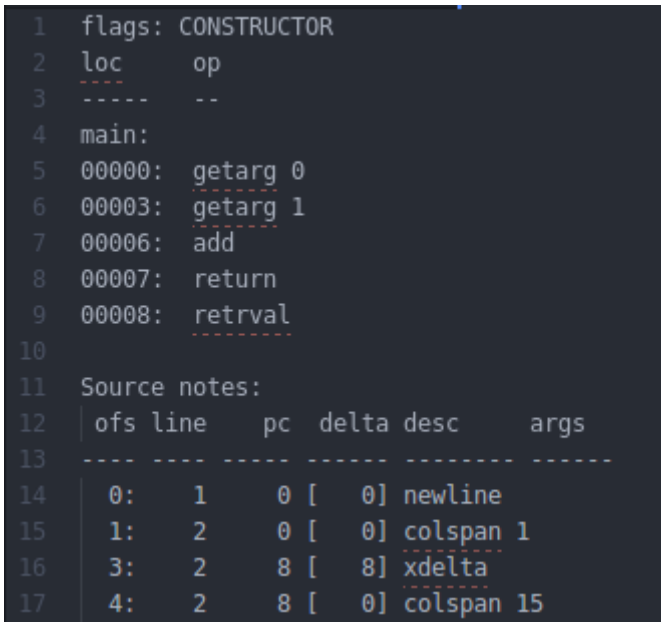


Uma vez que a abstract syntax tree (AST) é formada, com base no tipo da JavaScript engine, o

bytecode generator converte a AST para uma linguagem intermediaria ou código nativo, e isso é feito para cada bloco de código dentro da AST.

Bytecode, também conhecidos como **portable code**, são formar canónicas de representação de códigos que são projetados para obter execução eficiente por software interpretador.

Seguindo o passo de interpretação e compilação, abaixo podemos verificamos o bytecode gerado para a função “sum” que utilizamos como referencia:



```
flags: CONSTRUCTOR
loc     op
-----   --
main:
00000:  getarg 0
00003:  getarg 1
00006:  add
00007:  return
00008:  retval

Source notes:
 ofs line    pc  delta desc     args
---- ---- ----- ------ -------- ------
  0:    1     0 [   0] newline
  1:    2     0 [   0] colspan 1
  3:    2     8 [   8] xdelta
  4:    2     8 [   0] colspan 15
```

Após a geração do bytecode, entramos na fase de interpretação e execução da representação intermediária, o que é comumente feito de utilizando uma função simples, em vários passos, uma instrução por vez percorrendo o bytecode gerado.

Quando o bytecode chega até a area de execução, observamos a existência de dois componentes importantes para a performance de uma JS engine, que são o JIT (Just-in-Time) Compiler e o Garbage Collector.

O JIT, é responsável por traduzir bytecodes para código de maquina durante a execução do programa, ou seja, assim que um trecho de código é requisitado, o JIT transforma o bytecode para código de maquina, referente ao bloco solicitado e o disponibiliza para execução, fazendo com que a disponibilização do resultado seja rápida e não consuma ciclos de processamento desnecessários entregando somente os blocos que serão utilizados no momento em que forem requeridos.

Junto ao JIT, encontramos o Garbage Collector (ou coletor de lixo), que é a área responsável pelo gerenciamento automático da memória que é ocupada por objetos, sendo denominados como “lixo” as posições de memória que estão sendo ocupadas com informações referentes ao programa em execução mas que não são mais relevantes ou que não são mais utilizadas.

Tanto o JIT quando o Garbage Collector, são executados quando solicitados, e estão disponíveis durante toda a execução da JS engine, por isso estão diretamente ligados ao passo de execução dentro do esquema apresentado.

A JAVASCRIPT ENGINE SPIDERMONKEY

A JavaScript engine SpiderMonkey é um projeto de código aberto criado por Brendan Eich, na Netscape Communications em meados de 1996, e é considerada a primeira JavaScript engine da história.

Escrita utilizando linguagens de programação como C, C++ e JavaScript, pode ser executado em vários sistemas operacionais e devices como celulares, tablets e até mais recentemente TVs, o que a torna muito adaptável e com robustez.

ESTRUTURA DA JAVASCRIPT ENGINE SPIDERMONKEY

Por ser uma JavaScript engine, a SpiderMonkey possui a estrutura e segue um fluxo de interpretação de códigos JavaScript muito próxima ao apresentado no tópico ***Fluxo de funcionamento da JavaScript engine.***

Mesmo sendo uma JavaScript engine, sua estrutura contem algumas implementações que visam melhorar o desempenho na geração e interpretação de códigos JavaScript, bem como otimizar e melhorar o uso de memória durante sua execução.

O JUST-IN-TIME COMPILER NO SPIDERMONKEY

No SpiderMonkey, como uma forma de melhorar o desempenho durante a execução do JavaScript, existem duas camadas de JITs que funcionam separadamente e com objetivos diferentes.

A primeira camada, conhecida apenas como Baseline Compiler, e que tem como função apenas uma compilação preliminar, sem otimizações durante a execução, e foi introduzida para substituir a antiga camada method JIT chamada JaegerMonkey, com o objetivo de fácil manutenção e por possibilitar a otimização de novas funcionalidades na linguagem JavaScript.

A segunda camada, chamada de IonMonkey, que tem foi desenvolvido para

JAVASCRIPT SHELL (JS)

“The JavaScript shell (js) is a command-line program included in the SpiderMonkey source distribuction.”

- https://developer.mozilla.org/en-US/docs/Web/JavaScript/Shells

CONCLUSAO

- https://developer.mozilla.org/en-US/docs/Mozilla/Projects/SpiderMonkey/Hacking_Tips

- https://developer.mozilla.org/en-US/docs/Mozilla/Projects/SpiderMonkey/Parser_API

- http://www.scribd.com/doc/269666132/Work-With-JS-Engine-API

- https://www.quora.com/How-does-a-JavaScript-engine-work?share=1

- http://www.slideserve.com/oriana/v8-an-open-source-high-performance-javascript-engine

- https://www.quora.com/How-does-a-JavaScript-engine-work/answer/Amar-Prabhu

- http://static.googleusercontent.com/media/research.google.com/en//pubs/archive/37204.pdf

- http://static.googleusercontent.com/media/research.google.com/en//archive/papers/rajab-2011a.pdf

- http://static.googleusercontent.com/media/research.google.com/en//pubs/archive/36574.pdf

- http://www.slideshare.net/brunoborges/nashorn-nova-engine-javascript-do-java-se-8?qid=683d6d16-5fd1-4157-88ef-4936dac92217&v=default&b=&from_search=16

- http://www.slideshare.net/nwind/virtual-machine-and-javascript-engine?qid=683d6d16-5fd1-4157-88ef-4936dac92217&v=qf1&b=&from_search=2

- http://www.slideshare.net/shadedecho/javascript-architecture-the-front-and-the-back-of-it-3518425

- http://www.slideshare.net/axemclion/understanding-javascript-engines

- http://www.slideshare.net/RednaxelaFX/implement-js-krystalmok20131110

- http://www.slideshare.net/nwind/javascript-engine-performance?qid=683d6d16-5fd1-4157-88ef-4936dac92217&v=qf1&b=&from_search=4

- http://www.slideshare.net/RednaxelaFX/implement-js-krystalmok20131110?qid=683d6d16-5fd1-4157-88ef-4936dac92217&v=qf1&b=&from_search=6

- http://www.slideshare.net/lijing00333/javascript-engine?qid=683d6d16-5fd1-4157-88ef-4936dac92217&v=qf1&b=&from_search=7

- http://www.slideshare.net/amdgigabyte/know-your-javascript-engine?qid=683d6d16-5fd1-4157-88ef-4936dac92217&v=qf1&b=&from_search=8

- http://www.slideshare.net/senchainc/javascript-engines-under-the-hood?qid=683d6d16-5fd1-4157-88ef-4936dac92217&v=qf1&b=&from_search=12

- https://developer.mozilla.org/en-US/docs/Mozilla/Projects/Rhino/JavaScript_Compiler

- https://developer.mozilla.org/en-US/docs/Mozilla/Projects/SpiderMonkey/JSAPI_reference

- https://developer.mozilla.org/en-US/docs/Mozilla/Projects/SpiderMonkey/JSAPI_User_Guide

- https://wiki.mozilla.org/JavaScript:New_to_SpiderMonkey

- https://en.wikipedia.org/wiki/Just-in-time_compilation

- https://en.wikipedia.org/wiki/JavaScript_engine

- http://wiki.jvmlangsummit.com/images/c/ce/Nashorn.pdf

- https://en.wikipedia.org/wiki/List_of_ECMAScript_engineshttps://blog.mozilla.org/javascript/2013/04/05/the-baseline-compiler-has-landed/

- https://github.com/Benvie/continuum#continuum---a-javascript-virtual-machine-built-in-javascript

- https://en.wikipedia.org/wiki/Programming_paradigm#Multi-paradigm

- http://tratt.net/laurie/research/pubs/html/tratt__dynamically_typed_languages/

- https://en.wikipedia.org/wiki/SpiderMonkey_%28software%29

- https://blog.mozilla.org/javascript/page/2/

- https://blog.mozilla.org/javascript/2014/01/23/the-monkeys-in-2013/

-

- https://blog.mozilla.org/javascript/2012/08/22/hello-world/

- https://blog.mozilla.org/javascript/2015/02/26/the-path-to-parallel-javascript/

- http://www.umiacs.umd.edu/research/parallel/

BIBLIOGRAPHY

- SpiderMonkey (https://developer.mozilla.org/en-US/docs/Mozilla/Projects/SpiderMonkey)
- SpiderMonkey (software) (https://en.wikipedia.org/wiki/SpiderMonkey_%28software%29)
- SpiderMonkey Internals (https://developer.mozilla.org/en-US/docs/Mozilla/Projects/SpiderMonkey/Internals)
- Just-in-time compilation (https://en.wikipedia.org/wiki/Just-in-time_compilation)
- JavaScript Engine (http://en.wikipedia.org/wiki/JavaScript_engine)
- Dynamically typed languages, Laurence Tratt, Advances in Computers, vol. 77, pages 149-184, July 2009 (http://tratt.net/laurie/research/pubs/html/tratt__dynamically_typed_languages/)
- Dynamic programming language

(https://en.wikipedia.org/wiki/Dynamic_programming_language)

- JavaScript (https://en.wikipedia.org/wiki/JavaScript)
- JavaScript – The Definite Guide, Sixth Edition, David Flanagan, March 2011, O'Reilly
- Compilers: Principles, Technique, and Tools, Second Edition, Alfred V. Aho, Ravi Sethi, Jeffrey D. Ullman, Addison Wesley, 2007
- JSAPI User Guide (https://developer.mozilla.org/en-US/docs/Mozilla/Projects/SpiderMonkey/JSAPI_User_Guide)
- Introduction to the JavaScript Shell (https://developer.mozilla.org/en-US/docs/Mozilla/Projects/SpiderMonkey/Introduction_to_the_JavaScript_shell)
- SpiderMonkey Build Documentation (https://developer.mozilla.org/en-US/docs/Mozilla/Projects/SpiderMonkey/Build_Documentation)
- Parsing (https://en.wikipedia.org/wiki/Parsing)
- Abstract Syntax Tree (https://en.wikipedia.org/wiki/Abstract_syntax_tree)
- JavaScript AST visualizer (http://jointjs.com/demos/javascript-ast)
- What is JavaScript AST, how to play with it? (http://stackoverflow.com/questions/16127985/what-is-javascript-ast-how-to-play-with-it)
- How browsers work (http://taligarsiel.com/Projects/howbrowserswork1.htm)
- Bytecode (https://en.wikipedia.org/wiki/Bytecode)
- Bytecodes (https://developer.mozilla.org/en-US/docs/Mozilla/Projects/SpiderMonkey/Internals/Bytecodes)
- A LISP garbage-collector for virtual-memory computer system (http://dl.acm.org/citation.cfm?id=363280)
- Garbage collection (computer science) (https://en.wikipedia.org/wiki/Garbage_collection_%28computer_science%29)
- A parallel, real-time garbage collector (http://dl.acm.org/citation.cfm?id=378823)
- JSWhiz - Static Analysis for JavaScript Memory Leaks (http://static.googleusercontent.com/media/research.google.com/en//pubs/archive/40738.pdf)
- The Baseline Compiler Has Landed (https://blog.mozilla.org/javascript/2013/04/05/the-baseline-compiler-has-landed/)

-

REFERENCIAS

- Mozilla Source Code Directory Structure (https://developer.mozilla.org/en-US/docs/Mozilla_Source_Code_Directory_Structure)

- ECMAScript® 2015 Language Specification (http://www.ecma-international.org/publications/standards/Ecma-262.htm)

- What is the Document Object Model? (http://www.w3.org/DOM/#what)

- NodeJS (https://nodejs.org/)

ANEXO

- Mozilla Public License Version 2.0 (https://www.mozilla.org/MPL/2.0/)

- Duck Typing (https://en.wikipedia.org/wiki/Duck_typing)